論　文

バイアスの変更に対応できる学習戦略

正員　山田　太一[†]　　正員　犬塚　信博[†]　　正員　石井　直宏[†]

Learning Strategy Following Changes of Conceptual Bias

Taichi YAMADA [†], Nobuhiro INUZUKA [†] *and* Naohiro ISHII [†], *Members*

あらまし　帰納的学習問題では，学習問題を定義することによりその問題にバイアスが掛けられる．このバイアスは正しい解を得るために適切なものに定められる必要がある．しかし，学習問題を定義する段階で適切なバイアスを決めることは困難である．それを解決するために，本論文では学習を行いながらバイアスをより適切なものに変更することのできる学習戦略を提案する．本戦略は基本的枠組みとして学習戦略に有用なバージョン空間法を利用している．しかし，一般のバージョン空間法が保持する情報（仮説の集合 G と S）ではバイアスの変更に対応することはできない．そこで，本方法では，学習される概念に使用してはならない属性値の領域と使用される属性値の外延が包含しなければならない属性値の領域を記憶し，それによりバイアスの変更に対応する．本論文では，この情報のもち方について説明し，この情報からバイアスの変更に対応する方法を示す．更に，本アルゴリズムの正当性を示し，バイアスの変更に必要となる計算量の解析を行い，本方法の有効性を示す．

キーワード：帰納的学習，バージョン空間法，バイアス，アルゴリズム

1. まえがき

与えられた実例からそれを説明する概念を帰納的に学習する場合，対象空間，仮説空間，推論方法等を決める必要がある[(1)~(3)]．それにより無限の可能性から，得られる学習結果の枠が決まる．この枠のことをバイアスと言う[(3),(4)]．バイアスの決め方が正しくなければ，正しい結果を得ることはできない．従って，バイアスの決め方は重要である[(5)]．従来の方法[(1),(2)]では，このバイアスは学習前に決められ，学習途中にそれを変更することはできない．しかし実際には，学習前に決められたバイアスが最も適切であるかは保証されていないことが多い．もし，バイアスの変更が可能であれば，学習を行いながらより適切なバイアスを決めることができる[(4),(6)]．例えば，積分オペレータの適用条件を学習するSTABB[(4)]はシステム自身がバイアスの変更を行う．しかし，このシステムでは与えられた実例を記憶し再計算している．そこで本研究では，学習戦略として有用なバージョン空間法[(1)~(3)]を再計算せずにバイアスの変更に対応できる戦略に拡張した．

† 名古屋工業大学工学部電気情報工学科，名古屋市
Faculty of Engineering, Nagoya Institute of Technology, Nagoya-shi,466 Japan

バージョン空間法とは，次々と与えられる実例に無矛盾なすべての仮説の集合 H をその中で極大に一般的（以後，極大）な仮説の集合 G と極大に特殊（以後，極小）な仮説の集合 S により挟まれた領域（バージョン空間）として表現し管理する戦略である[(1)~(3)]．

この戦略では，求める概念に包含される正の実例と包含されない負の実例を与えることにより学習が進行する．正の実例が入力されると H がそれを包含するように S 内の仮説が一般化され，負の実例が入力されると H からそれを排除するように G 内の仮説が特殊化される．そして，S と G により表現されるバージョン空間 H が一つの仮説のみを含むとき，学習は終了する．

この戦略により学習を行うために必要となる実例や概念（仮説）を表現するための言語として，ここでは属性値の連言表現を使用する．そして，我々が対象とするバイアスの変更は，概念に使用できる属性値を加えるという変更である．これにより，概念の表現が豊かになりバイアスが弱められることになる．本論文では，学習途中にユーザがこの変更を行うことのできる戦略を提案する．

2. 例からの学習とバイアスの変更

実例や概念を表すのに必要となる特徴を属性と呼び，

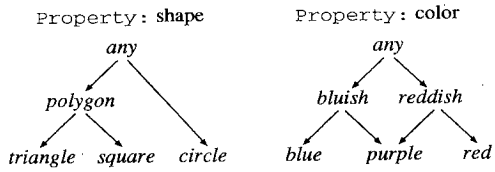

図1　属性グラフ
Fig. 1　Property graphs.

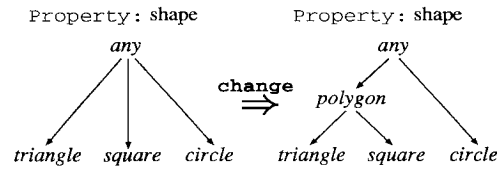

図2　属性の変更
Fig. 2　Change property.

その属性がとる値を属性値と呼ぶ．例えば，属性として色と形を考えれば，色という属性の属性値として赤や青があり，形という属性の属性値として三角や円がある．形式的には属性を属性値の集合として表現できる．例えば，形＝{三角, 四角, 円, 多角形, 任意} によって属性を表す．この属性（X とする）は次のような性質をもつ．

（1）　$x, y \in X$ 上に半順序関係 $\leqq$ が定義されている．

$y \leqq x$ のとき，x は y より一般的であるか等しい，あるいは，y は x より特殊であるか等しいと言う．

（2）　$\leqq$ に関して最大元（最も一般的な属性値）をもつ．

（3）　属性の中で一つの外延（属性値の集合）に対する属性値はただ一つである．

以後，一般に m 種類の属性を $X_1, \cdots, X_m$ で表す．

このような属性の属性値の関係を図1のようなグラフで表現する．これを属性グラフと呼ぶ．ここで，$a \rightarrow b$ のとき a は b より一般的である．

このとき，属性グラフの葉にあたる属性値はその属性の極小元（最も特殊な属性値）であり，根にあたる属性値が最大元である．

ここでは，実例記述言語（実例を記述する言語）として属性値の連言表現を使用する．例えば，図1の二つの属性において，“triangle∧red” のような属性値の連言で一つの実例を記述する．但し，実例に対して使える属性値は各属性の極小元（葉）のみである．つまり，実例空間は $I = X_1^L \times \cdots \times X_m^L$ である．但し，X_i^L は，属性 X_i の極小元の集合である．また，実例は $i \in I$ である．更にここでは，概念記述言語（概念を記述する言語）にも属性値の連言を使用する．但し，概念を表すためにはすべての属性値を使用できる．従って，概念空間は $Co = X_1 \times \cdots \times X_m$ である．そして，$c \in Co$ が概念である．ここで，$a \wedge b$ という表現は組 (a, b) に対応する．

このように定義された概念間には次の半順序関係が定義される．つまり，概念 $c_1 = (c_{11}, \cdots, c_{1m})$，概念 $c_2 = (c_{21}, \cdots, c_{2m})$ に対して次のとおり，

$$c_1 \leqq c_2 \Leftrightarrow (c_{1k} \leqq c_{2k}) \ (k = 1, \cdots, m)$$

但し，この半順序関係は，属性値の半順序関係と意味的に等しいので同じ記号 $\leqq$ を用いている．

また，実例も一つの概念であるが，特に，$i \leqq c$ の場合，概念 c が実例 i を包含し，$i \not\leqq c$ の場合，c が i を包含しない（排除する）と言う．

今，与えられた正の実例の集合を I^+，負の実例の集合を I^- とする．このとき，正の実例を包含し，負の実例を排除する概念の集合をバージョン空間 H と言い，次のとおりである．

$$H = \{c \in Co \mid \forall i^+ \in I^+ (i^+ \leqq c) \wedge \forall i^- \in I^- (i^- \not\leqq c)\}$$

また，このバージョン空間は次のような G と S によって挟まれた空間と考えることができる．つまり，

$$G = \{c_g \in H \mid \neg \exists c \in H (c_g \leqq c \wedge c_g \neq c)\}$$

$$S = \{c_s \in H \mid \neg \exists c \in H (c \leqq c_s \wedge c_s \neq c)\}$$

と書くことができる．逆に，H は次のように表せる．

$$H = \{c \in Co \mid \exists c_g \in G, \exists c_s \in S \ \ (c_s \leqq c \wedge c \leqq c_g)\}$$

つまり，G は H の中で極大な概念の集合であり，S は H の中で極小な概念の集合である．

このような学習に対してユーザが行えるバイアスの変更は図2により表される．この変更はある属性において，極小元の部分集合を外延とする新しい属性値を加えるという変更である．これにより，概念記述言語で使用できる言葉を増やすことができる．この変更は属性に対する変更である．従って，これを属性の変更と呼ぶ．この変更方法から明らかなように属性の変更はその属性の極小元を変更しない．また，属性のもつ性質3）から，その属性の最大元も変更しない．

3.　拡張されたバージョン空間法

本方法は，実例や概念を記述するための言語として複数属性を対象とする属性値の連言表現を使用しているが，アルゴリズムの本質は属性が複数であることには依存していない．しかし，実用性を考慮した場合，単数ではなく複数の属性を対象とする必要があり，また，単数属性から複数属性に拡張することは難しい．

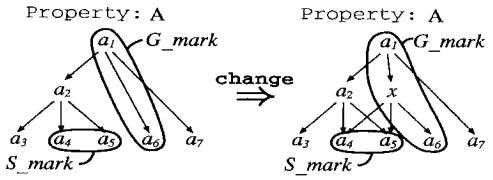

図3 *G_mark* と *S_mark*
Fig. 3 *G_mark* and *S_mark*.

そこで，アルゴリズムの本質的な部分は単一属性を対象に説明を行い，その詳細と証明については複数属性を対象に説明を行う．

3.1 アルゴリズムの本質

図3の属性 A の変更前の属性グラフを対象とする単数属性で学習を行うときを考える．

例えば，a_4 と a_5 という二つの正の実例と a_6 という負の実例が与えられたとする．このとき，G と S は，共に $\{a_2\}$ となる．これに対し本方法では，図3に示した *G_mark* と *S_mark* を記憶する．*G_mark* は目標概念に使用してはならない属性値の領域を表し，*S_mark* は目標概念に使用される属性値の外延が包含しなければならない属性値の領域を表している．この *G_mark*，*S_mark* から G と S を求めることができる．(3.2(4)参照)

ここで，図3に示した属性の変更を行ったとする．このとき新しく加えられた属性値 x は負の実例 a_6 を包含してしまう．従って，概念 x はバージョン空間には含まれない．本方法では，変更前の *G_mark* の中に x の子供にあたる a_6 が含まれていることからこのことを判断することができる．そこで，x が目標概念に使用できないことを記憶するために *G_mark* に x を加える．また，属性の変更は極小元を変更することはないので，*S_mark* を更新する必要はない．

このように *G_mark*，*S_mark* により属性の変更に対応する．また，この *G_mark*，*S_mark* から，与えられた仮説がバージョン空間に含まれるかどうかを判定する仮説チェックを行うことができる．つまり，チェックする仮説に使用されている属性値が *G_mark* の中に含まれず，*S_mark* の属性値をすべて包含していれば，バージョン空間に含まれる．

3.2 アルゴリズムに対する諸定義

(1) アルゴリズムが G の代わりに記憶する *G_mark*

複数属性の場合，負の実例の属性値のうち少なくとも一つが目標概念に矛盾する．そこで，与えられた負の実例に対しどの属性値を矛盾するものとみなしたか，その見方を node pattern (後述) で表しその集合を記憶する．

属性 X_j において負の実例として現れた属性値の集合を $N_j=\{i_j \mid (i_1,\cdots,i_j,\cdots,i_m)^- \in I^-\}$ で表す．

この N_j に対し，組 $(D_1,\cdots,D_m)$ (但し，$D_1 \subseteq N_1,\cdots,D_m \subseteq N_m$) を leaf pattern と呼ぶ．この leaf pattern に対して次のような関係を定義する．

$$(D_1,\cdots,D_m) \subseteq (D_1',\cdots,D_m')$$
$$\Leftrightarrow D_1 \subseteq D_1' \wedge \cdots \wedge D_m \subseteq D_m'$$

そして，leaf pattern l_1，l_2 に対して $l_1 \subseteq l_2$ のとき，l_1 は l_2 より包含してはならない属性値が少ない (制限が少ない) という意味で一般的であるという．

leaf pattern は属性の極小元 (葉) により構成されている．次に，その極小元を包含するすべての属性値を考慮した node pattern (後述) を考える．

$$General(x)=\{y \mid x \leqq y\} \quad (\text{但し，} x,y \in X)$$

$$General\_Set(D)=\bigcup_{d\in D} General(d)$$

としたとき，leaf pattern $(D_1,\cdots,D_m)$ に対し，$(General\_Set(D_1),\cdots,General\_Set(D_m))$ を node pattern と呼ぶ．この node pattern にも leaf pattern と同様の順序関係を定義する．このとき，*G_mark* は node pattern のある集合である．

(2) アルゴリズムが S の代わりに記憶する *S_mark*

複数属性の場合，正の実例として現れた属性値は目標概念が必ず包含する属性値である．従って，正の実例として現れた属性値を各属性に対して記憶する．

属性 X_j において正の実例として現れた属性値の集合を $P_j=\{i_j \mid (i_1,\cdots,i_j,\cdots,i_m)^+ \in I^+\}$ で表す．このとき，*S_mark* は次のように表される．

$$S\_mark=(P_1,\cdots,P_m)$$

(3) *G_mark*，*S_mark* が満たすべき条件

(1)，(2)で定義された *G_mark*，*S_mark* からバージョン空間 (G と S) が得られることを証明することができる．(3.4 参照) そのために *G_mark*，*S_mark* が満たすべき条件を以下に示す．

node pattern $n=(M_1,\cdots,M_m)$ に対して，次の命題が成り立ち，

$$\forall (i_1,\cdots,i_m)^- \in I^- (i_1 \in M_1 \vee \cdots \vee i_m \in M_m)$$

M_i に含まれる極小元が必ず負の実例として現れたものであるとき，n は I^- に対して無矛盾であると言う．また，*G_mark* の中のすべての node pattern が I^- に対して無矛盾であるとき，*G_mark* は I^- に対して無矛盾であると言う．

また，この定義は leaf pattern の集合に対しても同様に用いる．

[条件1] G_mark が I^- に対して無矛盾である．

node pattern $n=(M_1,\cdots,M_m)$ のある M_i に対して次の命題が成り立つとき，

$\forall x\in M_i(General(x)\subseteqq M_i\wedge\exists x^L\in M_i\cap X_i^L$
$(x^L\leqq x))$

M_i は pattern の無矛盾性を満足するといい，すべての $M_i\ (1\leqq i\leqq m)$ が pattern の無矛盾性を満たすとき，n は pattern の無矛盾性を満足するという．そして，G_mark の中のすべての node pattern が pattern の無矛盾性を満足するとき，G_mark は pattern の無矛盾性を満足すると言う．

つまり，この命題は M_i が $M_i\cap X_i^L$ に含まれる属性値を包含するすべての属性値からなる集合であることを示している．また，この定義から，明らかに $General(x)$ は pattern の無矛盾性を満足する．

[条件2] G_mark が pattern の無矛盾性を満足する．

G_mark が I^- に対して無矛盾な node pattern の中で極大なものをすべて包含しているとき，G_mark は含極大性を満足すると言う．また，この定義は leaf pattern の集合に対しても同じ意味で用いられる．

[条件3] G_mark が含極大性を満足する．

$S\_mark=(P_1,\cdots,P_m)$ に対し次の命題が成立し，

$\forall(i_1,\cdots,i_m)^+\in I^+(i_1\in P_1\wedge\cdots\wedge i_m\in P_m)$

```
main;
                          1        m
                          ∨        ∨
 1. G_mark ⇐ {(φ,...,φ)}  ;initialize G_mark
 2. S_mark ⇐ (φ,...,φ)    ;initialize S_mark
 3. while not(End)
 4.    case
 5.       (a)Learn
 6.       (b)Change Property
 7.       (c)Hypothesis Check
 8. end.

(a)Learn
 1. read instance i = (i_1,...,i_m)
 2. if "i is negative." then
 3.    G_mark' ⇐ φ
 4.    for (M_1,...,M_j,...,M_m) ∈ G_mark do
 5.       for j⇐1 to m do
 6.          G_mark' ⇐ G_mark'∪
                {(M_1,...,M_j ∪ General(i_j),...,M_m)}
 7.       fend
 8.    fend
( 9.  G_mark' ⇐ {n∈ G_mark'|n is maximal in G_mark'})
10. else ; "i is positive."
11.    S_mark' ⇐ (P_1 ∪ {i_1},...,P_m ∪ {i_m})
12.    where S_mark = (P_1,...,P_m)
(13.   G_mark' ⇐ {n∈ G_mark'|n is consistent with
 S_mark'})
14. end.

(b)Change Property
 1. j⇐property index
 2. x_new ⇐new property value
 3. child_set ⇐ {x ∈ X_j|x is child of x_new}
 4. Add x_new to X_j
 5. G_mark' ⇐ φ
 6. for (M_1,...,M_j,...,M_m) ∈ G_mark do
 7. if (M_j ∩ Child_set) ≠ φ
 8.    then G_mark' ⇐ G_mark'∪
             {(M_1,...,M_j ∪ {x_new},...,M_m)}
 9.    else G_mark' ⇐ G_mark'∪
             {(M_1,...,M_j,...,M_m)}
10. fend
11. end.

(c)Hypothesis Check
 1. read hypothesis C= (c_1,...,c_m)
 2. if (⋀_k(∀p ∈ P_k, p ≤ c_k))
 3.    ⋀(∃(M_1,...,M_m) ∈ G_mark, ⋀_k(c_k ∈ M_k))
 4.    then  write("C is consistent.").
 5.    else  write("C is inconsistent").
 6. end.
```

P_i に含まれる属性値が必ず正の実例として現れた属性値であるとき，S_mark は I^+ に対して無矛盾であるという．

[条件4] S_mark が I^+ に対して無矛盾である．

(4) G_mark，S_mark から得られるバージョン空間

G_mark の各 node pattern についてその node pattern がもつ属性値以外の属性値の中で極大な属性値を用いた概念の集合を G' とする．つまり，

$G'=\{(c_1,\cdots,c_m)\mid\exists(M_1,\cdots,M_m)\in G\_mark,$
c_j は X_j-M_j の極大値，$j=1,\cdots,m\}$

また，各属性に対して S_mark がもっている属性値をすべて包含する属性値の中で極小な属性値を用いた概念の集合を S' とする．

このとき，G' の中で極大で S' のある概念より一般的である概念の集合が G となり，S' の中で極小で G' のある概念より特殊である概念の集合が S となる．

3.3 アルゴリズムの説明

本論文で提案するアルゴリズムを図4に示す．但し，( )の付いた操作((a)9, 13)は G_mark から不必要な要素を取り除くためのものであり，このアルゴリズムのもつ本質的な操作には無関係である．従って，ここではこの操作を行わないものとしてアルゴリズムの説明を行う．

このアルゴリズムを説明するために，図5に示され

図4 アルゴリズム
Fig. 4 Algorithm.

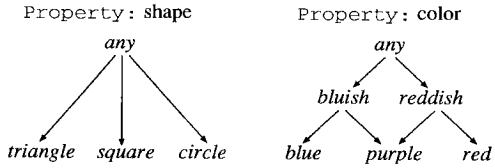

図5 属性グラフ
Fig. 5 Property graphs.

る属性グラフを考える．

初期化のアルゴリズムにより，*G_mark*，*S_mark* が初期化された後，負の実例 (*square*, *red*) と二つの正の実例 (*triangle*, *blue*)，(*square*, *purple*) を順に入力したとする．各実例に対して(a)Learn (学習) を選択し，*G_mark*，*S_mark* が次のように更新される．

$G\_mark = \{(\{any, square\}, \phi), (\phi, \{any, reddish, red\})\}$

$S\_mark = (\{triangle, square\}, \{blue, purple\})$

ここで，**3.2**(4)で述べた方法で G'，S' を作り，それに対して G と S を求めると次のように目標概念が得られた状態となる．

$G' = \{(triangle, any), (circle, any), (any, bluish)\}$

$S' = \{(any, bluish)\}$

$G = \{(any, bluish)\}$

$S = \{(any, bluish)\}$

ここで，図2で示した属性の変更を行うとする．(b)Change Property (属性の変更) を選択し，*G_mark*，*S_mark* が次のように更新される．

$G\_mark = \{(\{any, polygon, square\}, \phi), (\phi, \{any, reddish, red\})\}$

$S\_mark = (\{triangle, square\}, \{blue, purple\})$

同様にして，G と S を求めると次のようにバージョン空間が広げられたことがわかる．

$G = \{(any, bluish)\}$

$S = \{(polygon, bluish)\}$

ここで，(c)Hypothesis Check (仮説チェック) を選択し，ある仮説が与えられた実例に矛盾するかどうかのチェックを行うとする．仮説 (polygon, bluish) を入力すれば，(c)2，3の条件が成立し *consistent* が表示される．しかし，仮説 (*polygon*, *any*) を入力すれば，(c)3が不成立となり *inconsistent* が表示される．

3.4 アルゴリズムの正当性

3.4.1 バージョン空間の保存

まず，アルゴリズムが保持する *G_mark* と *S_mark* から，**3.2**(4)により得られる G と S が与えられた実例に対するバージョン空間の G と S となることを示す．

[定理1] 条件1，条件2，条件3，条件4が成立するならば，**3.2**(4)により得られる G と S が与えられた実例に対するバージョン空間の G と S となる．

(証明)

(I) *G_mark* のもつ node pattern は，与えられた負の実例についてどの属性値を矛盾するものとみなしたかという見方を表している．従って，条件1，条件2，条件3が成立すれば，G' は与えられた負の実例をすべて排除できる概念の中で極大な概念をすべて含む．

(II) *S_mark* のもつ属性値は与えられた正の実例として現れたものであり，目標概念が必ず包含する属性値である．従って，条件4が成立すれば，S' は与えられた正の実例をすべて包含できる概念の中で極小な概念をすべて含む．

(I)，(II)より，G' と S' から得られる $G(\subseteq G')$ と $S(\subseteq S')$ はまさしく与えられた実例に対するバージョン空間の G と S である． □

次に実際のアルゴリズムの逐次的な操作により得られる *G_mark*，*S_mark* に対して常に各条件が成立することを示す．但し，ここでは操作(a)9，13を行わないときを考える．

[定理2] アルゴリズムの実行中，常に条件1，条件2，条件3，条件4が成立する．

これを示すために，まず次の補題1を示す．但し，負の実例の集合 I^- に対して無矛盾な node pattern のうち極大な node pattern の集合を I^- に対する max pattern set と呼ぶ．

[補題1] *G_mark* が I^- に対する max pattern set (*MP_Set* とする) を含むとき，操作(a)4～8により，*G_mark'* の中に $I^- \cup \{i^-\}$ に対する max pattern set (*MP_Set'* とする) が含まれる．

(補題1の証明) max pattern set の定義より，*MP_Set'* に含まれる node pattern が *MP_Set* の中のある node pattern よりも特殊か等しいことは明らかである．従って，*MP_Set'* は，*MP_Set* の中の各 node pattern に対して次の性質 (無矛盾極大性) をもつ node pattern の和に包含される．そこで，アルゴリズムが実際にこの和を計算していることを示す．

[無矛盾極大性] もとの node pattern より特殊か等しく，$I^- \cup \{i^-\}$ に対して無矛盾である node pattern のうち極大であり，pattern の無矛盾性を満足する．

ある $n \in G\_mark$ に対して操作(a)5～7により *G_mark'* に加えられる node pattern の集合を T とす

れば，この T が n に対して無矛盾極大性を満足するものをすべて包含していることを示せばよい．ここで，T は次のようになる．但し，$n=(M_1,\cdots,M_m)$ とする．

$T=\{(M_1,\cdots,M_j\cup General(i_j),\cdots,M_m)\mid 1\leqq j\leqq m\}$

今，n に対して無矛盾極大性を満足する $\tilde{n}=(\tilde{M}_1,\cdots,\tilde{M}_m)$ を考える．このとき，$n\subseteqq\tilde{n}$ である．よって，$\tilde{n}$ が $I^-\cup\{i^-\}$ に対して無矛盾で極大となるためには，たかだか一つの k で次式が成り立てばよい．

$$i_k\in\tilde{M}_k\wedge M_k\subseteqq\tilde{M}_k \qquad (1)$$

なぜなら，式(1)を満たす k が二つ以上ある node pattern は，そのうち一つの k で式(1)が成り立つ node pattern よりも特殊となり，$\tilde{n}$ の極大性に反する．よって，$\tilde{n}$ に対してたかだか一つの k で式(1)が成り立つ．このとき，そのような k において，$M_k\cup General(i_k)=\tilde{M}_k$ が成り立つことを示す．

$M_k\cup General(i_k)\neq\tilde{M}_k$ と仮定する．

もし，$M_k\cup General(i_k)-\tilde{M}_k\neq\phi$ ならば，$M_k\cup General(i_k)$，$\tilde{M}_k$ の満たす pattern の無矛盾性と $i_k\in\tilde{M}_k$ から，$M_k\nsubseteqq\tilde{M}_k$ となる．これは式(1)に矛盾する．

また，もし，$\tilde{M}_k-M_k\cup General(i_k)\neq\phi$ ならば，$M_k\cup General(i_k)$，$\tilde{M}_k$ の満足する pattern の無矛盾性と式(1)から，$M_k\cup General(i_k)\subseteqq\tilde{M}_k$ となる．これは $\tilde{n}$ が極大であることに反する．

これらより，$M_k\cup General(i_k)=\tilde{M}_k$ となり $\tilde{n}$ は T に含まれることになる．従って，T は確かに無矛盾極大性をもつ node pattern をすべて包含している．

(補題1の証明終)□

(定理2の証明) 実例の個数および，属性変更の回数についての帰納法で証明する．

(I) 実例の入力も属性の変更もないとき

G_mark，*S_mark* が各条件を満足することは明らか．

(II) 実例が入力されているとき

この状態で次に実例の入力，属性の変更を行ったとき，各条件が保存されることを示す．但し，区別するために更新された *G_mark*，*S_mark* をそれぞれ *G_mark'*，*S_mark'* とする．

(II-1) 負の実例 $i^-=(i_1,\cdots,i_m)$ が入力されたとき，操作(a)6から *G_mark'* が $I^-\cup\{i^-\}$ に対して無矛盾であることは明らかであり，条件1が成立する．

$General(x)$ が pattern の無矛盾性を満足することから，操作(a)6において *G_mark'* に加える node pattern $(M_1,\cdots,M_j\cup General(i_j),\cdots,M_m)$ は pattern の無矛盾性を満足する．従って，*G_mark'* に対し条件2が成立する．

G_mark に対し条件3が成立するので *G_mark* は I^- に対する max pattern set をもつ．このとき補題1から，*G_mark'* は $I^-\cup\{i^-\}$ に対する max pattern set を含む．従って，*G_mark'* に対し条件3が成立する．

また，このとき *S_mark* に対する処理は行わないので，*S_mark'* に対し条件4が成立する．

(II-2) 正の実例 $i^+=(i_1,\cdots,i_m)$ が入力されたとき，*G_mark* に対する操作は行わない．よって，*G_mark'* に対し条件1，条件2，条件3が成立する．

操作(a)11より *S_mark'* は $I^-\cup\{i^+\}$ に対して無矛盾である．従って，*S_mark'* に対し条件4が成立する．

(II-3) ある属性 X_j に対して属性の変更を行ったとき，*S_mark* に対する処理は行わないので，*S_mark'* に対し条件4が成立することは明らかである．

属性変更の定義から，M_j の極小元は変わらない．従って，*G_mark'* に対し条件1が成立する．

G_mark' に対して条件2が成立することを示すために，操作(b)8の $M_j\cup\{x_{new}\}$ と操作(b)9の M_j が，pattern の無矛盾性を満足することを示す．

①操作(b)8が行われる場合

このとき $(M_j\cap Child\_Set)\neq\phi$ である．$(M_j\cap Child\_Set)=MC$ とおく．M_j は pattern の無矛盾性を満たすので，MC の中の属性値より一般的な属性値はすべて M_j に含まれる．よって，x_{new} より一般的な属性値はすべて M_j に含まれる．また，MC の中の属性値は M_j の中のある極小元よりも一般的である．従って，その親である x_{new} も M_j の中のある極小元よりも一般的である．よって，$M_j\cup\{x_{new}\}$ は pattern の無矛盾性を満足する．

②操作(b)9が行われる場合

このとき $(M_j\cap Child\_Set)=\phi$ である．つまり，x_{new} の子供の中で M_j に包含されるものはない．従って，x_{new} は M_j に無関係である．よって，(b)9の M_j は pattern の無矛盾性を満足する．

①②より，*G_mark'* に対し条件2が成立する．

また，属性の変更は極小元を変更しない．従って，*G_mark'* に対し条件3が成立する．

(II-1)，(II-2)，(II-3)より，実例が入力された状態で各条件が保存されることが示された．

従って，(I)(II)より，アルゴリズムは各条件を保存する．

(定理2の証明終)□

[定理1][定理2]より，このアルゴリズムは常にバージョン空間を保存していることが示された．

3.4.2 仮説チェックの正当性

ここでは，仮説チェックのアルゴリズムにより，与えられた仮説がバージョン空間に含まれるかどうかを正しくチェックできることを示す．

条件4の成立より，与えられた仮説が *S_mark* の属性値をすべて包含するならば(アルゴリズム(c)2)，その仮説は正の実例に対して無矛盾である．しかし，包含できなければ矛盾する．また，条件1，条件2，条件3の成立より，*G_mark* の中にその仮説の属性値を一つももたない node pattern があれば((c)3)，その仮説は与えられた負の実例に対して無矛盾である．しかし，なければ矛盾する．従って，これらの条件((c)2，3)により，仮説がバージョン空間に含まれるかどうかを判定できる．

3.5 情報の削減

前節では，アルゴリズムにおいて *G_mark* に対する情報の削減((a)9，13)を行わないときを考えた．そこで，本節では情報の削減を行ってもバージョン空間を正しく求められることを示す．

(1) *G_mark* の中で極大でない node pattern を削除する．((a)9)

操作(a)9は次式で表される．

$$G\_mark' = \{n \in G\_mark' \mid \neg \exists n' \in G\_mark'(n' \subsetneqq n)\}$$

まず，負の実例の入力によって *G_mark* が更新されたとき，*G_mark* が極大な node pattern のみを保持していれば，条件1，条件2，条件3の成立を保存できることを示す(正の実例の入力が行われたときには *G_mark* は変更されないため条件を保存できる)．

条件1，条件2は *G_mark* の中の各 node pattern に対する条件であるから，極大な node pattern がこれらの条件を満足し，実例の入力に対してもこれらの条件を保存することは明らかである．

また，補題1から，負の実例が入力されたとき，*G_mark* が極大な node pattern を保持していれば，更新された *G_mark'* に対し条件3が成立する．

次に，属性の変更が行われたときを考える．このとき，負の実例を入力したときと同様の理由から，条件1，条件2は保存される．また，属性の変更は変更する属性の極小元を変えることはない．従って，条件3が成立することは明らかである．

よって，*G_mark* において極大でない node pattern を削除しても条件1，条件2，条件3は保存される．また，*S_mark* は *G_mark* の影響を受けないため条件4も成り立つ．従って，バージョン空間を正しく得ることができる．

(2) *G_mark* の中で *S_mark* との関係で矛盾する node pattern を削除する．((a)13)

ここで，*G_mark* の中で *S_mark* との関係で矛盾する node pattern とは，次の命題を満たす node pattern である．但し，node pattern を $(M_1, \cdots, M_m)$，*S_mark* を $(P_1, \cdots, P_m)$ とする．

$$\exists j(M_1 \cap P_1 \neq \phi \vee \cdots \vee M_j \cap P_j \neq \phi \vee \cdots \vee M_m \cap P_m \neq \phi) \quad (2)$$

ある j でこの条件を満たす node pattern $n=(M_1, \cdots, M_m)$ のもつ属性値以外の属性値を用いて概念を作れば，そこから得られる概念は $M_j \cap P_j$ の属性値を包含できない．つまり，この概念はバージョン空間の中には含まれない．

次に，ある状態で式(2)を満足する node pattern n が *G_mark* の中にあるとする．このとき，アルゴリズムの実行により n が更新されても n のもつ属性値が削除されることはない．従って，式(2)を満足する node pattern から得られる node pattern は必ず式(2)を満たす．そして，このような node pattern から得られる概念はバージョン空間に含まれることはない．

よって，式(2)を満足する node pattern を *G_mark* から削除してもバージョン空間は常に保存される．

4. アルゴリズムの計算量

本研究が提案したアルゴリズムとバージョン空間法との計算量の比較を行った．但し，計算を簡略化する

表1 アルゴリズムの計算量

	本アルゴリズム	バージョン空間法
正の実例入力時	$O(mc^h g_m)$	$O(mp^{h-1}s^2 + mp^{h-1}gs + mp^{2h-2}s)$
負の実例入力時	$O(m^2c^h g_m^2 + mc^{2h}g_m^2 + mc^h p^{h-1}g_m + mp^{2h-2}g_m)$	$O(mp^{h-1}g^2 + mp^{h-1}sg + mc^h p^{h-1}g)$
仮説チェック時	$O(mc^h p^{h-1} + mc^h g_m)$	$O(mp^{h-1}s + mp^{h-1}g)$
属性の変更時	$O(c^{h+1}g_m)$	$O(P(mp^{h-1}s^2 + mp^{h-1}sg + mp^{2h-2}s) + N(mp^{h-1}g^2 + mp^{h-1}sg + mc^h p^{h-1}g))$

ために以下の変数を定めて解析を行った．結果を表1に示す．

属性数：m
属性グラフの高さ：h
各属性値の親の個数：p 以下
各属性値の子供の個数：c 以下
バージョン空間の G の要素数：g
バージョン空間の S の要素数：s
G_mark の要素数：g_m
入力された正の実例の個数：P
入力された負の実例の個数：N

g と g_m はオーダでは変わらず（ともに $0(2^{mch})$）また，p と c もそれほど差はない．従って表1より，正の実例の入力時には，本方法の方が計算量は少なく，負の実例の入力時には，本方法の方が計算量は多いことがわかる．これは，本方法では属性の変更を考慮したために，G_mark が複雑な情報のもち方をしていることが原因であると考えられる．しかし，実用上問題のある差ではない．また，仮説チェック時にはそれほどの差はない．それに対して，属性の変更時には，バージョン空間法では与えられた実例に対して学習をやり直さなければならないのに対し本方法では，実例の個数によらず少ない計算量で対応できる．これは，バージョン空間法に比べ本方法が属性の変更に対して有効であることを示している．

5. むすび

本研究では，バージョン空間法をバイアスの変更に対応できるように拡張した．従来のバージョン空間法ではこのような変更に対応するには再計算の必要があるのに対し，この方法ではその必要がない．しかし，属性の変更を考慮するため node pattern のような複雑な情報のもち方をしており，その分記憶量が増加している（グラフの属性値の個数倍程度）．従って，この情報のもち方を工夫する必要がある．また，本アルゴリズムではバージョン空間が空となるときの判定を行っていないため，その判定方法を求める必要がある．更に今後，その空になった状態から与えられた実例に無矛盾な仮説が存在するように（バージョン空間を空でなくなるように）属性の変更を自動的に行えるアルゴリズムに拡張し，バイアスの学習を試みたい．

文　献

（1） Mitchell T. M. : "Generalization as Search", Artif. Intell., **18**, pp. 203-226 (1982).
（2） Mitchell T. M. : "Version spaces : An approach to concept learning", Ph. D. Thesis, Stanford University (Dec. 1978).
（3） 仁木和久，岩崎　俊："概念の帰納的学習"，人工知能学会誌，**3**, 6, pp. 695-703 (1988).
（4） Utgoff P. E. : "Machine learning of inductive bias", KLUWER ACADEMIC PUBLISHERS (1986).
（5） Russel S. J. and Grosof B. N. : "Change of Representation and Inductive Bias", pp. 267-308, KLUWER ACADEMIC PUBLISHERS (1990).
（6） Rendell L., Seshu R. and Tcheng D. : "LAYERD CONCEPT-LEARNING AND DYNAMICALLY-VARIABLE BIAS MANEGEMENT", Proc. IJCAI, **10**, 1, pp. 308-314 (1987).
（7） 山田太一，犬塚信博，石井直宏："概念記述言語の変更に対応できる学習戦略"，信学技報，**AI90**-69 (1990).

（平成3年5月20日受付，10月2日再受付）

山田　太一

平2名工大・工・電気情報卒．現在，同大大学院博士前期課程電気情報工学専攻在籍．帰納的学習に関する研究に従事．

犬塚　信博

昭62名工大・工・情報卒，平1同大大学院博士前期課程電気情報工学専攻了．現在，同大大学院博士後期課程電気情報工学専攻在籍．人工知能，特に帰納的学習に関する研究に従事．

石井　直宏

昭38東北大・工・電気卒．昭43同大大学院博士課程電気および通信工学専攻了．同年同大・医・助手．昭50名工大・工・助教授を経て，現在，同大・電気情報工学科教授．この間，しきい値論理，医用情報処理，および非線形処理の研究に従事．工博．